SRC

A MODEL OF CONCEPT LEARNING

RICHARD H. SHERMAN

1. This document has been approved for public release and sale ; its distribution is unlimited.

SRC-68-8

A MODEL OF CONCEPT LEARNING

RICHARD H. SHERMAN

SRC-68-8

SYSTEMS RESEARCH CENTER
CASE WESTERN RESERVE UNIVERSITY
UNIVERSITY CIRCLE
CLEVELAND, OHIO 44106
JUNE, 1968

This work was sponsored in part by the USA Air Force Office of Scientific Research under Grant # AF-AFOSR-125-67, and by the National Science Foundation under Grant # GK-1386. The author was given support during this study by an NDEA Fellowship.

A MODEL OF CONCEPT LEARNING

Richard H. Sherman

ABSTRACT

Concept learning is investigated by constructing a computer program named CE, which is similar to EPAM in many respects. The main feature of this research is the learning of hierarchial concepts, i.e., concepts in terms of other concepts. In addition the concepts can be either conjunctive or disjunctive. CE can generalize on the instances given to it, and it can adapt, i.e., it can correct errors made in generalization. The learning ability of CE is demonstrated on two example tasks: a concept learning task, and a rote learning task. Computer experiments confirm the effectiveness of CE on these example tasks. Although not simulated on the machine, we investigate a geometry analogy task which requires relational concepts.

ACKNOWLEDGMENTS

It gives the author great pleasure to acknowledge the continuous help and valuable suggestions of Dr. George Ernst.

The research reported here was supported in part by Airforce Grant # AF-AFOSR-125-67, NSF Grant # GK-1386.

The author was given support during this study by an NDEA Fellowship.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER I

INTRODUCTION

Concept learning is an important complex information process. As such it has received the attention of scientists who study the representation and processing of information. Three such groups of scientists are philosophers, psychologists and computer scientists. Our approach is a mixture of the approach of psychologists and computer scientists, our objective is artificial intelligence - the construction of computer porgrams that exhibit intelligence. The purpose of Section 2.3 is to place our approach into its' proper perspective.

Before we can do this we need a definition of concept learning. This is given in Sections 2.1 and 2.2. Our definition is sufficiently general to include work in pattern recognition. The difference between work in pattern recognition and concept learning is mainly one of approach. Some workers (e.g., Banerji [2]), consider "concept learning" and "pattern learning" to be the same process.

The purpose of this thesis is twofold. The first is to discover a "better" model of concept learning. One important feature of this model is the ability to generalize. Another feature of this model is the ability to learn concepts in terms of other concepts.

The second purpose is to investigate the relationship between concept learning and a rote learning theory called EPAM, which is also a computer program. A computer program that is similar to EPAM in many respects has been constructed. The program is called Concept-EPAM, CE, to distinguish it from other versions of EPAM. Due to the similarity of CE to EPAM we give an explanation of EPAM in Section 2.5.

The basic difference between EPAM and CE is the introduction of the set membership relation, ϵ; EPAM only uses the equality relation, =. The usefulness of this new relation is described in Chapter III by exploring how concepts are stored in memory. This extension of EPAM changes both the retrieval process, described in Chapter III, and the learning process, described in Chapter IV.

The learning strategies of CE are two interacting processes, image elaboration and tree modification. The hierarchial structure of CE is apparent in the discussion of image elaboration.

The implementation of CE is described in Chapter V. The experiments that have been given to CE are described in Chapter VI. After some basic information is given to CE, two tasks are performed: a concept learning task, and a pair associate task. Although not simulated on the machine, we describe a geometry analogy task which requires relational concepts.

Chapter VII contains conclusions and observations.

CHAPTER II

APPROACHES TO CONCEPT LEARNING

The main purpose of this chapter is to describe other research in concept learning so as to place this work in its' proper perspective. To accomplish this we first give some definitions.

2.1 The Definition of the Task

A concept is a set of objects that are members of some universe. A description of a concept is an expression in some language which is satisfied by every member in the concept. Every concept has a description, however, the "size" of the expression may be quite large. Concept learning is the process of generating a description of a concept given some objects that are in the concept and other objects that are not in the concept.

The procedure used to investigate concept learning is the same one that is used in a psychological experiment. The subject, CE in our case, is presented with exemplars* and is told whether or not they are members of a concept. After acquiring a description of the concept the subject is presented with other exemplars, some of which he has never seen, and asked if they are in the concept. This is basically the same task used by Hunt [8].

* Exemplars are "typical" members of concepts. This terminology is from Bruner [4]; for the most part we use his terminology.

Information is presented to CE in the form of a triplet, <exemplar> <name> <sign>. For example, the input ABC C1 +, means that the three letter word ABC is an exemplar of the concept C1 and CDE C1 - means that CDE is not a member of concept C1.

Information is retrieved from CE by inputing the triplet, <exemplar> <name> ?. An output of + or - occurs depending on whether the object (exemplar-name pair) is found to be true or false. The question, <exemplar> ? + , will retrieve the name if the exemplar is contained in only one concept. For example, we would like ABC ? + to retrieve the name C2. However, most exemplars are in several different concepts, as we will see. Examples of the retrieval process are presented in Chapter VI.

Generalization is the ability to predict whether an object never seen before is a member of a concept. Since only a few members of a concept are given, generalization is necessary to learn the concept. Generalization from a list of given elements cannot be perfect. A contradiction to the generalization can appear at any time when new lists of elements are exhibited. The predictive ability of a concept learner is "good" if correct identification occurs most of the time. A concept learner must be able to correct mistaken generalization when necessary. We are not concerned with avoiding mistakes altogether but with "good" generalization and with the adaptive abilities of the concept learner.

2.2 Classes of Concepts

Several types of concepts are now defined. The simplest case is the conjunctive concept. A conjunctive concept is represented by the conjunction of the features possessed by all of the exemplars in the concept. For example, the concept, C1, of those three letter words whose first letter is A and whose second letter is B, is a conjunctive concept. We can denote it by AB_ C1. The dash indicates that the third letter is irrelevant. ABC and ABD are both exemplars of C1.

A disjunctive concept is the union of several conjunctive concepts. For example, the concept, C2, of those three letter words whose first letter is A or whose first letter is B, is a disjunctive concept. This concept can be represented by the disjunction of two conjunctive concepts, (A _ _ ∨ B _ _) C2. The dash again indicates irrelevant letter positions and the wedge indicates logical "or."

A hierarchial concept defines sets of objects whose parts are also concepts. Let us consider the concept 'A' _ _ C3, where 'A' stands for any written letter a, and C3 is the concepts' name. ABC, abc, *abc* , etc., are all members of C3 because the first letter of each is an a. If the disjunctive, 'A', of all written a's is learned, C3 becomes just a simple conjunctive concept, 'A' . This illustrates the advantage (or necessity) of describing one concept in terms of another.

A relational concept is a set of objects, each object has two (or more) subparts that satisfy some relation. For example, the following objects have subparts which are related by "next"; ACB, BEC, and FAG. This relational concept can be described as "the third letter is 'next' after the first letter." Geometry analogy problems are another example of objects whose subparts are related. These problems are discussed in Chapter VI.

2.3 Related Work in Concept Learning

Concept learning is a fundamental process according to philosophers. For example, Quine [12] considers concepts as the basis for understanding language. Quine [12] states "Conceptualization on any considerable scale is inseparable from language, and our ordinary language of physical things is about as basic as language gets."[1] Quine [12] examines the notion of meaning and objective reference. However, he is more concerned with the structure of language then the process of acquiring concepts.

The most popular approach, typified by Sebestyen [13], is the evaluation of discriminant functions. Statistical techniques are often used to determine coefficients of the discriminant function. Although the number of coefficients that need to be determined are large, some good techniques have been developed for determining them. This approach is called "classification theory." A more powerful method called feature extraction is desireable. Feature extraction is a process that generates "good features." Feature extraction is a difficult problem and few pattern recognizers incorporate it, rather they are dependent on the initial measurements (i.e., basic features). The purpose of feature extraction is to reduce the number of terms in the discriminant function. The only work known to us which incorporates feature extraction is Uhr and Vossler [16]. Their

[1] Willard Quine, Word and Object (Cambridge Press, 1960) p.3.

features are templets, and a few pertinent templets are selected from a set of 75 templets.

A classical study in concept learning is by Bruner, Goodnow, and Austin [4]. They define concepts and deal mainly with conjunctive concepts, but they do discuss disjunctive concepts. Several strategies for learning concepts are proposed. The strategy for learning disjunctive concepts only allows for objects whose subparts have binary values. Experiments on human subjects, similar to the type performed by CE, are used to test the theories of learning. Bruner's subjects learn concepts made up of basic concepts well known to humans, e.g., number of boarders on a card, shapes, etc. Bruner [4] does not investigate how humans learn these basic concepts from sense data. Bruner's strategies are not used in CE, at least not in an obvious way.

The work of Hunt [8] relates a computer model of concept learning with research on concept learning in humans. He notes that concept learning "... is a hierarchial affair. One must have a concept in order to learn more concepts."[2] Hunt does not deal with learning "concepts of concepts." The issue of hierarchial concepts is primarily how this research differs from Hunt's. Concepts of concepts are built into the basic structure of CE.

Banerji [1] also noticed the usefulness of concepts of concepts. Banerji's work is concerned with expressing objects and concepts in

[2] Hunt, Marin, Stone, *Experiments in Induction*, (Academic Press, N.Y. and London, 1966), p. 12.

a formal description language. Essentially our description of objects do not differ from his, both languages contain the relations, set membership, ε, and equality, =. However, Banerji's language is more general, in particular, the language described in Banerji [1] contains quantification of variables. A concept learning algorithm for one of Banerji's languages was proposed by Windeknecht [17]. The algorithm was successful; however, it applies only to a restricted class of concepts.

Our approach combines Banerji's description language and EPAM. The language of CE is similar to Banerji's and the structure of CE is similar to EPAM. We describe EPAM next and following that (Chapter III) we describe CE.

2.4 Rote Learning and Concept Learning

Rote learning in humans is studied in an experiment called pair-associate learning. In this experiment the subject is presented with a list of paired terms. The first term of the pair is called the stimulus and the second term is called the response. After being presented with the list, the subject is shown a random stimulus and is asked to reply with the appropriate response. After his reply the correct response is given. The list is then recycled through until the subject answers correctly to each stimulus on the list.

The basic problem facing the subject is to learn to make a discrimination between different items on the list. The phenomenon called stimulus generalization occurs in which the subject reacts with the same response to different stimuli. Thus after learning a response to a particular stimulus, a whole class of stimuli will elicit the same response.

The idea of stimulus generalization forms a connection between experiments in rote learning (in particular EPAM since it exhibits this phenomenon) and experiments in concept learning. In the pair-associate task a response to one stimulus may be elicited by other stimulus which have not appeared on the original training list. Bower and Trabasco [3] have suggested an experimental continuum from the pair-associate task, in which each stimulus is paired with a unique response, to a concept learning task, where several stimulus are paired with the same response.

The process of generalization is not as simple as we seem to indicate. There are influences which restrict the range of generalization and restrain the subject from making the same response to all physically similar stimulus. The manner in which humans draw boundaries between members of a class and non-members of a class is not easily understood. The relation between generalization which occur in humans in performing the pair-associate and concept learning tasks is not obvious. Hunt states:

> The fact that list length has different effects in the concept learning and the pair-associates task, taken together with the fact that generalization between stimuli appears to play a different part in each situation, suggests that different psychological processes are involved and that one simulation ought not to be expected to cover both fields.3

CE uses the same processes for both tasks.

[3] Ibid., p. 193.

2.5 EPAM

EPAM is capable of learning (memorizing) objects. The objects used in CE are the same as those used in EPAM. Objects are either simple or compound. A *simple object* is a list of pairs, the first member is an attribute; the second member is either *1* or *0*. All simple objects have the same attributes. A *compound object* is a list of pairs, the first member is an attribute; the second member is an object, either simple or compound. The *universe* of objects contains all the objects possible for given attributes.

An example of a simple object is the letter, A, which can be encoded as,

$$((f_1\ 0)\ (f_2\ 1)\ \ldots).$$

The f_i's are the attributes of simple objects. An example of a compound object is the "nonsense syllable," XAT which is encoded as,

$$((1^{st}\ X)\ (2^{nd}\ A)\ (3^{rd}\ T)).$$

Here 1^{st}, 2^{nd} and 3^{rd} are attributes paired with the simple objects *X*, *A*, *T*, respectively.

Before we explain how EPAM learns these objects, it is helpful to discuss the type of information processing model with which we are concerned. The model is shown in Figure 1. The information processor receives input information from the universe and stores information about them in an internal representation. Information about the universe of objects can then be retrieved by asking questions of the system. Feedback from the universe is employed by

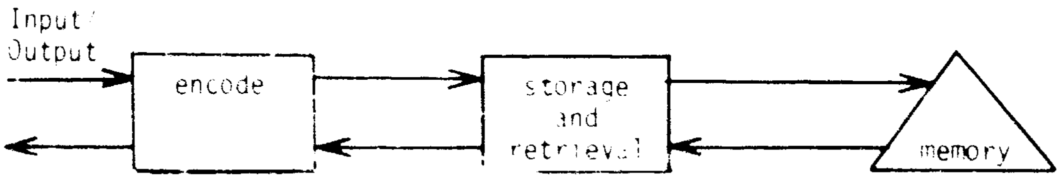


Figure 1

The EPAM Information Processing System

analyzing responses to these questions in relation to the correct responses. Internal changes are then made to decrease the number of responses in error. The changes that occur in the internal structure are called the processes of learning. The types of changes that occur for any particular situation depend on the strategies for learning.

The input is encoded and then sent to the memory. The output is received from memory and transmitted to the environment. The objects described above are outputs of the encoder. This thesis is mainly concerned with the memory and the storing and retrieving algorithms of the concept learning system.

The CE model of learning has basically the same structure as does EPAM. The organization and processes of CE is different and is presented in Chapter III.

The structure of EPAM is a binary decision tree. The non-terminal nodes contain tests. When an object enters the binary tree it follows a path down to a terminal node. Its' path through the tree depends on the result of the node tests. If an object passes a test, the "+" branch is taken, if the test fails, the "-" branch is followed. At the terminal node is stored an internal image of an object that comprises all the information memorized by the learner about that object. Associated with the partial image is a cue token. When an object reaches a terminal node, the cue token stored at that node evokes a new image from the discrimination net in response. This process of evoking a response is called the association process.

The discrimination learning process is made up of two interacting mechanisms called image building and tree modification. The image building mechanism elaborates the internal image stored at the terminal node. The tree modification mechanism grows the discrimination net by constructing a subnet of tests based on differences found and appending the new subnet at the terminal node. An example of the learning process in EPAM on a pair-associate experiment is now shown.

Let us assume that EPAM already has been presented with a list of pair-associate non-sense syllables. Figure 2 shows a snapshot picture of a typical EPAM discrimination net. The terminal node is _A'_ where the blanks indicate that no information is stored about the first and third letter positions, and the prime means that only a partial image of an A is stored. The cue token at the terminal node is R'E'P'. This cue token points to the response REP also stored in the tree but not shown in Figure 2.

Suppose the following two pair-associate nonsense syllables are presented to the net:

Stimulus	Response
XAZ	REP
XAQ	ZOX

The first stimulus object XAZ is sorted to the terminal node shown in Figure 1. The cue token now evokes the stored response, and REP is outputed. The output is compared with the paired response

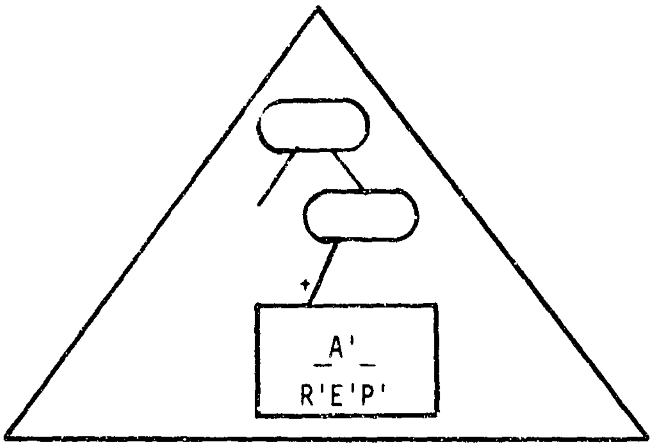


Figure 2

A Snapshot Picture of an EPAM Tree

and found to be correct. The partial image is now elaborated by storing new information, it becomes X'A'_.

Next the input XAQ sorts to the same terminal node, the response REP is again given. This is a demonstration of stimulus generalization since different stimuli produce the same response. However, when the output is compared with the paired term, ZOX, an error is registered. To correct the error, discrimination between the object XAQ and the partial image occurs by modifying the net. A test on some difference is added to the EPAM tree and a new image is formed. The new tree is shown in Figure 3. Note that the response image Z"O"X" is also stored in the discrimination net and a cue, Z'O'X', is stored with the object.

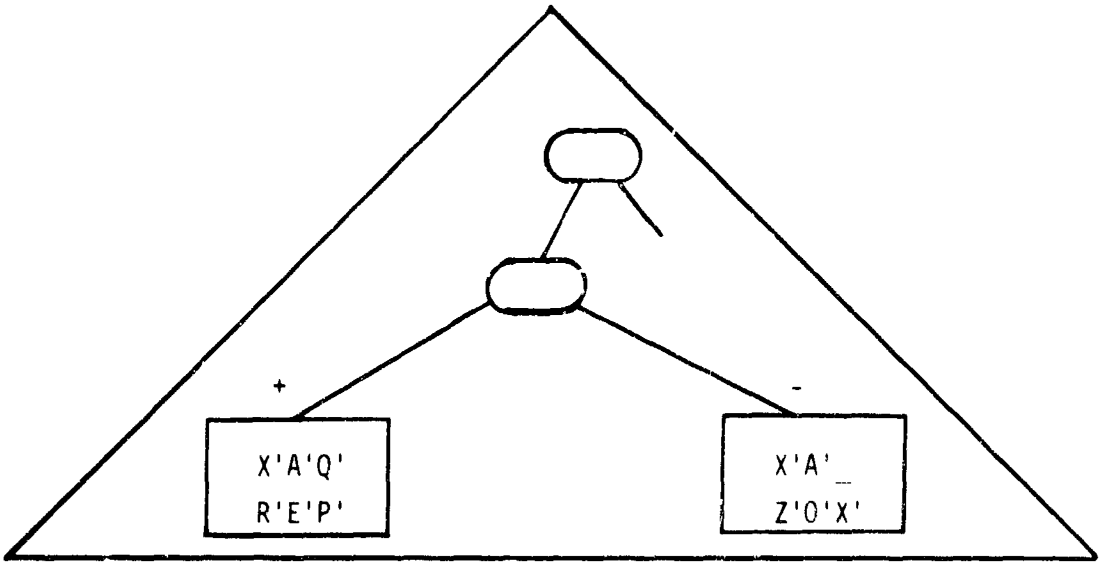


Figure 3

EPAM Trec After Learning Additional Stimuli

CHAPTER III

ORGANIZATION OF CE'S MEMORY

EPAM is capable of storing or memorizing discrete objects both "simple" and "compound." In CE only classes of objects are stored. Each class of objects represents a concept. A terminal node represents the conjunction of all objects that can be sorted to the node and match the image at the node. Thus, conjunctive concepts are similar to partial images in EPAM. Since a conjunctive concept is located at one terminal node, many different objects will sort to the same terminal node. This implies that the strategy for learning has to be different than that used by EPAM, because we do not want to discriminate between the objects of a single concept. The learning strategy is discussed in the next chapter.

A disjunctive concept is represented in the CE discrimination net by letting two or more terminal nodes have the same name. For example, the disjunctive concept C2, described by the two conjunctive concepts A   C2 and B   C2 is represented at two terminal nodes in Figure 4. Both the name (C2) and a description of a conjunctive concept (e.g., A   ) are stored at a terminal node. If we ask the question, ACD C2 ? (i.e., is ACD contained in the concept, C2), the object, ACD C2, will be sorted to the left node of the CE tree shown in Figure 4. Since the object matches the

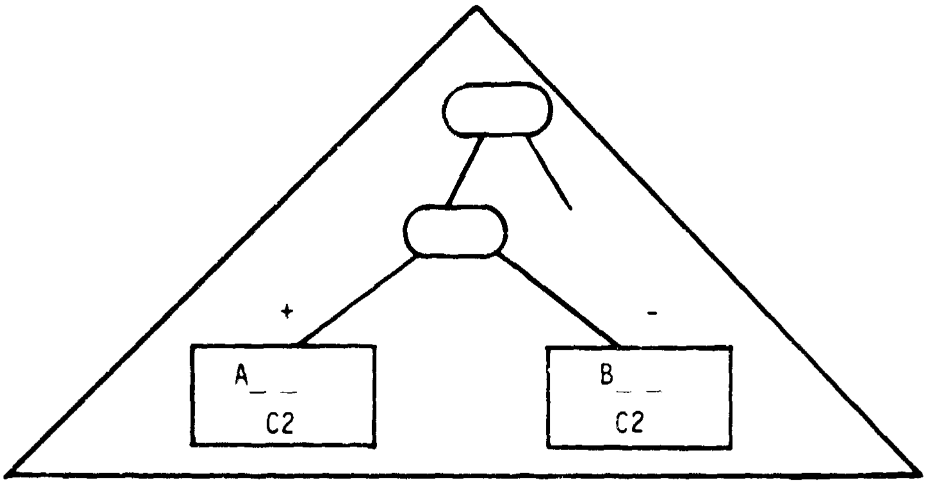


Figure 4

A CE tree is shown. The boxes are terminal nodes. The lines represent arbitrary paths to the terminals.

image, the answer to the question is +. The object BCD C2 would sort to the right terminal node in Figure 4, and again the answer is +. Thus, C2 is the disjunctive concept, A_ _$_{\vee}$B_ _.

The hierarchial concept 'A'_ _ C3, where 'A' stands for any written letter a, is a simple description of the disjunctive concept A_ _$_{\vee}$a_ _$_{\vee}$$a$_ _. Naturally, we want C3 to occupy a single terminal node. Therefore, some non-terminal node must be the test "is the exemplar's first letter a written a" (i.e., (ex 1^{st}) ϵ 'A'). If the encoding of A and a are sufficiently different, the test (ex 1^{st}) = a would turn C3 into a disjunctive concept as shown in Figure 5. The object, abc C3, would fail the test (ex 1^{st}) = A while, ABC C3, would pass the test. Hence, the requirement that C3 (and other concepts of the same form) occupy a single terminal node implies that a non-terminal node may be a set membership test instead of an equality test. The generalization to set membership tests is quite complex. For example, to answer the test (ex 1^{st}) ϵ 'B' we might "throw" the first letter, say b, of an exemplar, back into the tree. But b may be in many different concepts, e.g., lower case letter, vowel, etc. We must assume that concepts "overlap" with each other in a complex way (i.e., the intersection of two concepts is not necessariiy empty). However, we can answer, a ϵ 'A' because it is either true or false. This is not a paradox; it implies that there must be tests on names. Thus, objects are stored in the tree with the possibility of tests on names.

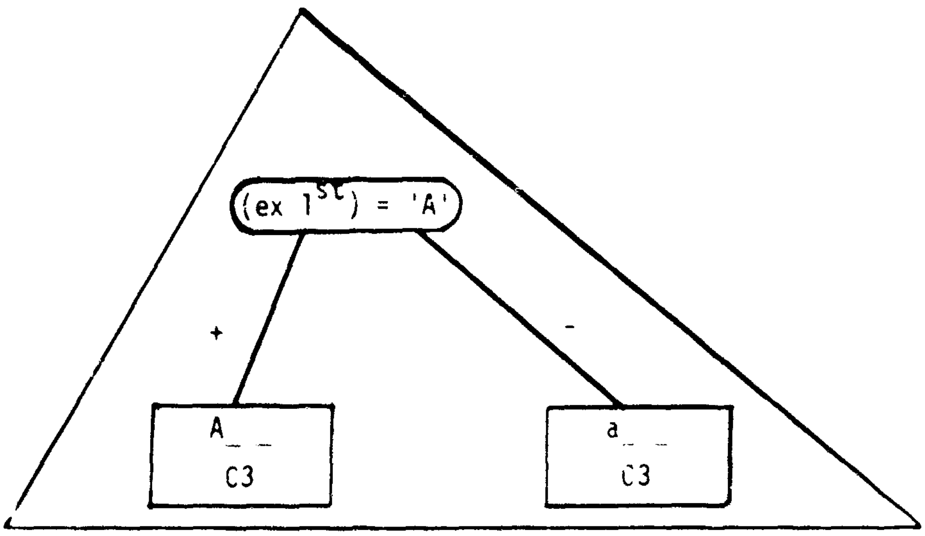


Figure 5

A tree with one equality test and two terminal nodes is shown.

Due to the importance of tests, we define the class of tests that can be used at non-terminal nodes of a CE tree. A test has the form, <feature> <relation> <image>. The relation is either set membership or equality. The feature is an ordered list of attributes that designates a subpart of an object. The image is the information stored at some terminal node when the feature does not designate part of a "simple" object (otherwise, the image is either '1' or '0' and the relation must be equality).

To explain how tests are performed on objects, we briefly give an example of a compound object in CE. The compound object, ABC C2, is encoded as a pair:

(1) (ex ((1st A) (2nd B) (3rd C)))

and

(2) (na ((1st C) (2nd 2)))

The attribute ex denotes that the second member of (1) is an exemplar; na denotes that the second member of (2) is a name. The second member of (1) is also a compound object. A,B,C,2 are simple objects and may be encoded, e.g., as

(3) $A = ((f_1\ 1)\ (f_2\ 0)\ (f_3\ 1)\ \ldots)$

f_1, f_2, f_3, ... denote elementary attributes of letters, i.e., number of straight line segments, arcs of circles, etc., and 0, 1 are values.

A feature of a test designates a subpart of an object. The first attribute of a feature designates an immediate subpart, X, of the object. The second attribute of the feature designates an immediate subpart of X, etc., and the last attribute designates the subpart for which the test is applicable. Suppose that the test, ((ex 1^{st}) ϵ 'A') is applied to ABC C2 as encoded above. <u>ex</u> designates the second member of the pair (1) and 1^{st} designates the <u>A</u> in (1). Thus, in this case, the test becomes A ϵ 'A' whose truth value depends upon what has been learned, previously.

Now we can state algorithm 1, used for the evaluation of the set membership test. (Equality tests are similar to the EPAM tests.)

1. Find the subpart designated by the feature of the test.
2. Form the question ⟨subpart⟩ ⟨image⟩ ? and answer it as if it were an input question.

As an example of algorithm 1 suppose the input ABC C2 + sorts to the non-terminal test node (ex 1^{st}) ϵ 'A'. The question A 'A' ? is thrown back into the tree. If a terminal node is found that matches the object A 'A', the test (ex 1^{st}) ϵ 'A' is true, otherwise false.

The images at terminal nodes are now explained. An <u>image</u> is the conjunction of several tests. Each test has the same form as a test at a non-terminal node. However, in addition to = and ϵ, the relation can be $\neq$ or $\notin$ (the negation of = and ϵ, respectively). The usefulness of these relations is discussed in the next chapter. An example of an image is the conjunction of the following tests:

$$\{\ ((\text{ex } 1^{st}) \in A'),$$
$$((\text{ex } 2^{nd}) \in B'),$$
$$((\text{na}) = (C3)'\ )\ \}.$$

The object ACD C3 matches this image because the statements $A \in A'$, $C \in B'$, and $C3 = (C3)'$ are all true. The prime on the letter $\underline{A}$ in the first test indicates that A' is an image of $\underline{A}$. Thus, $\underline{A}'$ is not the same as the externally encoded $\underline{A}$, but only some partial image of $\underline{A}$. Note an object is said to match an image when all tests from the image on the object are true. An object is different from an image if any test of the image is false on the object.

Names of concepts as well as exemplars are represented as concepts in CE. The name C3 is different every time it is written, i.e., C3 c3, C3, are all slightly different. Thus, the name is really a word whose first letter is a $\underline{C}$ (maybe capital, Roman) and second letter is a 3. Due to the special role of names we assume that names can be learned as disjoint conjunctive concepts.

Names are also learned in the tree. When an input sorts to a new terminal node the name is thrown back into the tree and learned as a concept, unless the name is an atomic symbol used for output. This atomic symbol is called a print name and its' use is shown in Section 6.

An example of learning a name is now given. When the object ACD C3 reaches a new terminal node (a node with no image) the name C3 is inputed into the tree as the list (OBJ. (1^{st} C)(2^{nd} 3)) where OBJ. indicates that the encoding is a single object; C and 3 are the simple objects that are subparts of an input object. The test ((na) = (C3)') is then formed at the new terminal node. The image (C3)' is identical with the image of the name C3 stored at an individual node.

To execute the test ((na) = (C3)') on an object, the feature na is found on the object. Then the object subpart remaining is matched with (C3)'. If the object subpart matches the image, then the test has passed, otherwise the test has failed.

If the input is an image (such cases arise from algorithm 1) the subpart na on the input is itself an image. The na image on the input cannot be compared with the test image since descriptions cannot be compared. However, we can require all tests in the test image, say (C3)', to be possessed by the subpart na image on the input for the test to be passed, otherwise the test has failed.

CHAPTER IV

LEARNING STRATEGY

As in EPAM learning consists of two processes: 1) Constructing and elaborating images at terminal nodes of the net, and 2) Growing the net.

First we discuss the image elaboration process, then we describe how the CE tree is modified by learning.

4.1 Image Elaboration

The learning strategy applies to objects of any complexity. (Complexity is the depth of nesting of compound objects.) If the subparts of a compound object are learned in the CE tree as a member of a concept, then the compound object should be easier to learn. Learning must then involve throwing subparts of objects back into the tree to retrieve information. (Image elaboration of simple objects does not involve throwing subparts back.) Suppose, for example, the input BBC C3 + is sorted to a new terminal node and we want to form an initial focus* image. The exemplars immediate subparts are thrown into the tree to retrieve information, i.e., the question, B ? +, is asked. Since this is a question no learning of the object will take place.

*This usage of 'focus' is the same as in Bruner [4].

There are three possibilities in retrieving information.

1. The subpart, B, is a member of only <u>one</u> concept, 'B'. Then the name 'B' is retrieved.

2. In general we have "overlapping" concepts and B will be a member of many concepts. Thus, tests on names will be encountered in sorting, B ? +, to a terminal node. When a test on the name is encountered, a search of the tree (i.e., an investigation of both branches leading from the test on the name) is used to find some names. If a terminal node is found such that B matches the image of the exemplar, then the image of the name is retrieved.

3. If the subpart, B, is not in any concept, we throw the subparts of B back to retrieve information about B itself. In this case B is a simple object and no subparts can be thrown back into the net. Chapter VI will give an example of this process on more complex exemplars.

To summarize, image elaboration of an object $\underline{X}$, involves searching the CE tree for those concepts of which some subpart of $\underline{X}$ is a member. If no immediate subpart of $\underline{X}$ is a member of a previously learned concept then the subparts are considered. Search arises because some object may be in many different concepts.

4.2 Modifications of the CE Tree Due to Learning

Learning only occurs when information is presented to CE; not when questions are asked. In CE the first image at a terminal node serves as a focus image for concept learning. When additional objects are sorted to a terminal node image adjustments are made or the net is grown by adding a new test and a new terminal node. To illustrate the strategy for net modification, several cases will be examined. The different cases depend on the three variables: 1) the sign of the input (+ or -), 2) the result of comparing the exemplar of the input with the tests at a terminal node, 3) the result of matching the name of the input and the image of the name at the terminal node. The cases are summarized in Table 1; we give examples below.

Case 1: The input, ABC C3 +, is sorted to a terminal node whose image is {((ex 1^{st}) ε A'), ((na) = (C3)')}. In this case, no modification occurs because no difference is detected. Both tests on the exemplar, ABC, and on the name, C3, are true. Thus, the input has already been "learned."

Case 2: The input, ABC C3 +, is sorted to a terminal node whose image is {((ex 1^{st}) ε D'), ((na) = (C3)')}. In this case, a difference is detected so we mark the test irrelevant, which we denote by placing an X next to the test, i.e., ((ex 1^{st}) ε D' X).

If the input, ASC C3 +, is sorted to the terminal node whose image is ((ex 1^{st}) ε A'), ((na) = (C3)'), then mark the test ((ex 1^{st}) ε A') irrelevant since it is false on the input exemplar.

Case Number	Name Match	Exemplar Match (tests on exemplar)	Input Sign	Action
1	matches	true	+	do nothing
2	matches	false	+	make test X irrelevant or grow tree
3	matches	all tests true	-	make all tests Y irrelevant or erase Y irrelevant on false test
4	matches	false	-	do nothing
5	different	all tests true	+	grow tree on name test
6	different	false	+	grow tree on one difference
7&8	different	true or false	-	do nothing

Table 1

A Summary of Tree Modifications

Case 3: The input, ABC C3 −, is sorted to a terminal node whose image is ((ex 2^{nd}) = B'), ((ex 1^{st}) = A'), ((na) = (C3)');. All exemplar tests are true on the input so we mark all the tests irrelevant by placing a Y next to the tests. The new node is now {((ex 1^{st}) = A' Y), ((ex 2^{nd}) = B' Y), ((na) = (C3)')}. The reason for using a different irrelevance sign in Case 3 from the sign used in Case 2 is explained after the sample cases.

If the input ABC C3 −, is sorted to a terminal node whose image is

((ex 1^{st}) = A'), ((ex 2^{nd}) = D' Y), ((na) = (C3)');

then the action is to erase the Y sign on the test ((ex 2^{nd}) = D'). In this example all exemplar tests are true, however, there is a test marked Y irrelevant which is false on the input.

Case 4: The input, ABC C3 −, is sorted to a terminal node whose image is {((ex 1^{st}) = D'), ((na) = (C3)')}. The test on the exemplar is false and we do nothing in this case

Case 5: The input is ABC C3 + and the terminal node is ((ex 1^{st}) = A'), ((na) = (C4)'. Since all tests on the exemplar are true, the tree is grown by using the test on the name, ((na) = (C4)').

Case 6: The input is ABC C3 +, and the image at the terminal node is ((ex 1^{st}) = D'), ((na) = (C4)') The test on both the exemplar and the name are false, and the tree is grown by using a test on the exemplar, i.e., ((ex 1^{st}) = D').

Case 7&8: A negative input, ABC C3 -, is sorted to a terminal node and the name of the input does not match the name at the node. Nothing is done to the node in these cases.

When all tests at a node are marked irrelevant, the node is considered blank and image elaboration occurs. Thus, Table 1 only shows cases of tree modification.

One additional case of tree modification is not shown in Table 1. If the input arrives at a terminal node where all tests are marked irrelevant and in addition no new tests are found by means of image elaboration, then CE assumes the concept to be a disjunctive concept. For example, if the input, ABC C3 +, sorts to the terminal node

$$\{((\text{ex } 1^{st}) \in A' \ X),\ ((\text{ex } 2^{nd}) \in D' \ X),\ ((\text{ex } 3^{rd}) \in D' \ Y),$$
$$((\text{na}) = (C3)')\}$$

and no new tests are found by means of image elaboration, a new non-terminal tree test is formed from one of the irrelevant tests marked with an X. The criteria for determining which test is best illustrated by means of an example given in Chapter V.

A few descriptive remarks about CE tree modification are now made. For conjunctive concepts, a feature possessed by a positive instance of a concept but not possessed by another positive instance of the same concept is always irrelevant to the concept. This type of irrelevance occurs in Case 2 of Table 1 and is denoted by the sign, X. Another kind of irrelevance, denoted by the sign Y, occurs in Case 3. When a negative instance of a concept possesses a feature that a positive instance also possesses, then that feature is only *possibly* irrelevant to the concept.

Learning of concepts in CE involves a process of changing the tests at the terminal node image. When the image has many tests, the extent of the concept is small because only a few input exemplars match the image. When the image has only a few tests, the extent of the concept is large because many input exemplars match the image. If a positive input does not match an image, then the extent of the concept is increased by making tests irrelevant. This action corresponds to Case 2 of Table 1. If a negative input matches an image as in Case 3 of Table 1, then the extent of the concept is decreased by making previously irrelevant tests or by making all tests irrelevant thus forcing image elaboration to occur.

CHAPTER V

THE IMPLEMENTATION

The main reason for programming CE is to evaluate its performance on actual tasks. In general, when a few simple processes (e.g., CE image elaboration and net modification processes) interact in a complex way the result may be unpredictable. Another purpose of coding the program is to demonstrate that the theory of concept learning presented in this report is sufficiently complete and detailed to be implemented on a digital computer. The computer program provides a check on errors in the theory of CE. These errors can be simple omissions of details, simple logical errors, or substantial omissions of pertinent theory. Chapter III and Chapter IV give a sufficient statement of CE's theory; the precise CE model is only stated in the working program.

CE is coded in a machine language called Sleuth II on the Univac 1107 computer at Case Western Reserve University. List processing facilities, (see Ernst [5]), consist of a number of subroutines which can be referenced from the Sleuth II language. Ernst [5] shows how the description lists of IPL and the property lists of LISP can be converted into type lists. The objects of Section 2.5 (used in CE) can easily be encoded into the type lists of Ernst [5]. The reason for choosing typed-lists to represent data, besides being available, is to combine qualities of both lists and description lists into a single data structure.

The top level organization of CE is shown in Figure 6. The executive runs the experiment and as such is a non essential part of the program. The executive is described in the first section of this chapter. The memory is described extensively in Chapter III and will not be further discussed in this chapter. In Section 5.2 the storage and retrieval algorithms are discussed.

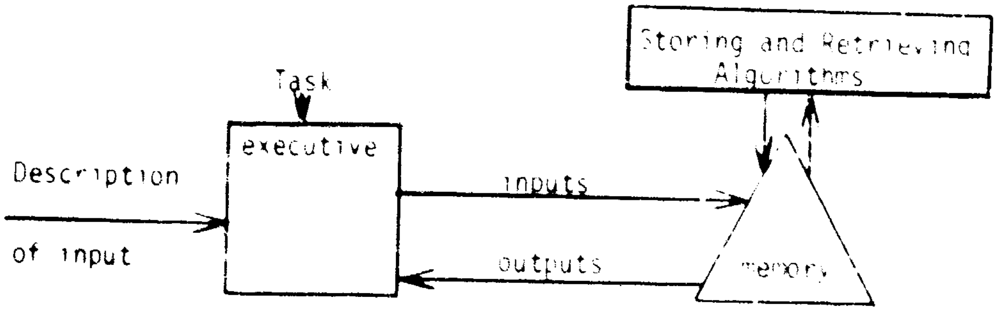


Figure 6

The Top Level Organization [illegible]

5.1 The Executive

The executive controls the flow of inputs to memory. Attribute-value pairs on objects are given to the executive. Inputs are then generated by augmenting the given descriptions with random noise at the lowest level (i.e, simple objects). For example, a list of relevant feature values (e.g., 8 out of 15 features are relevant) of a B is used as a description. Remaining feature values are then added at random to produce a typical B.

The storing and retrieving algorithms are techniques for processing information. In CE the rote learning task requires all inputs to have a '+' sign while for the concept learning task inputs have '-' signs also. The executive has the flexibility of monitoring both types of tasks.

5.2 Storing and Retrieval

The CE program is written to process inputs of arbitrary complexity. The program subroutines must be able to employ the same processes on data of arbitrary depth in their list structures. This is accomplished by recursively calling the subroutine within itself depending on the level of the list structure.

The general flow of control in the CE program is diagramed in Figure 7. To understand the sequence of operations we shall follow an input through the flow chart. The input is processed as follows:

1. The first node in the tree is located.
2. The node may be a test (non-terminal node) in which case control is transfered to step 3. If the node is a terminal node control transfers to step 5.
3. The node is a test. The test is executed on the input object. Because of algorithm 1 a question may be asked of the tree in executing this test. This question, constructed by a test in the tree, is called an internal question as opposed to an external question asked by the experimenter. The internal question is thrown back into the tree to step 1, recursively. The yes-no answer is then received and control continues to step 4.
4. The next node is found and control goes to step 2.
5. The node is a terminal node. The input is matched with the terminal node. Depending on the sign of the input,

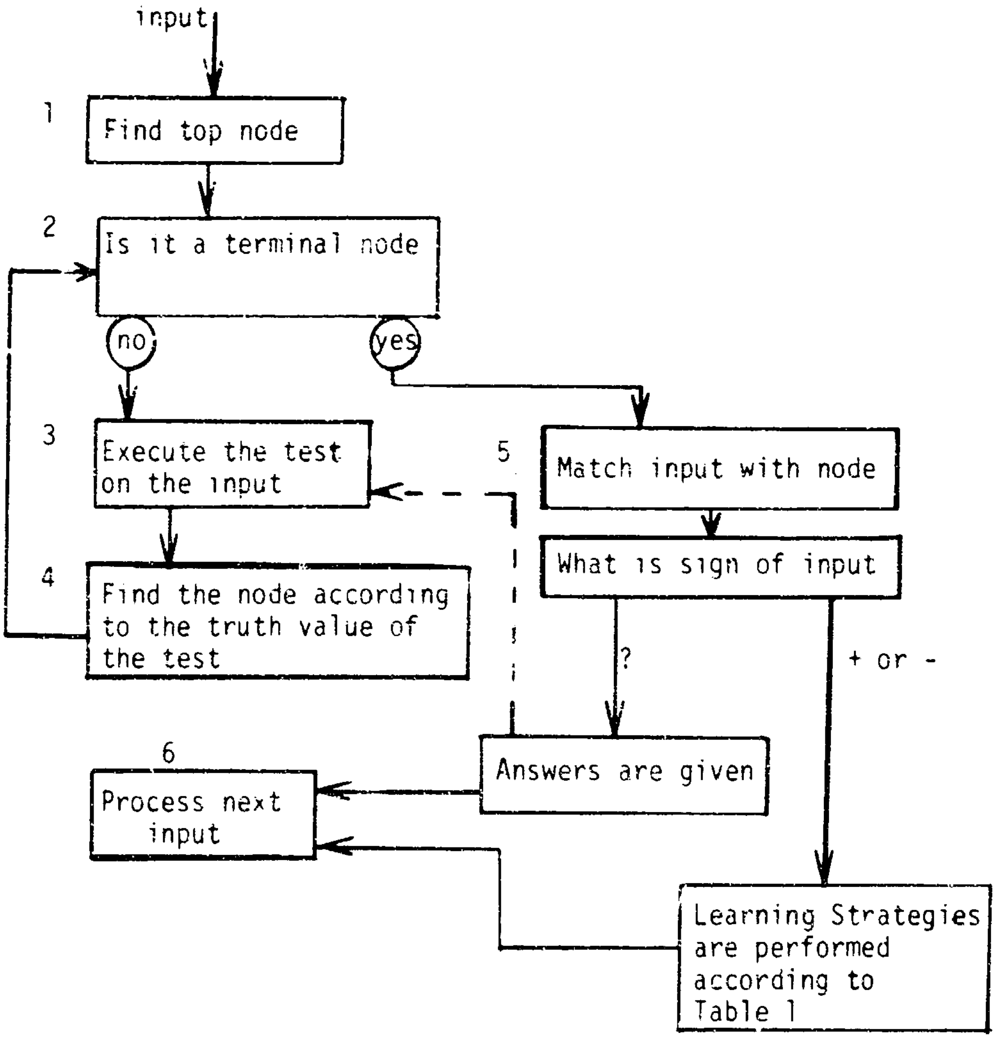

input
1
Find top node
2
Is it a terminal node
no
yes
3
Execute the test on the input
4
Find the node according to the truth value of the test
5
Match input with node
What is sign of input
?
+ or -
Answers are given
6
Process next input
Learning Strategies are performed according to Table 1

Figure 7

General CE Flow Chart

control may be transfered to one of the following places:

a. If the input is an internal question then the yes-no answer is given according to the match at the node. Control is returned to the recursive test in step 3.

b. If the input is an external question, asked by the experimenter, it may be one of two types. If the input is a yes-no question, <exemplar> <name> ?, then the output is given and control exits to step 6. If the question is to retrieve the name of an exemplar, <exemplar> ? +, the name at the terminal node is thrown back into the tree recursively to retrieve the individual name. The name is then outputed and control exits to step 6.

c. If the input object is information presented by the experimenter to the tree, control passes to the learning strategies. Image elaboration or tree modification (Table 1) occurs depending on the match at the terminal node.

6. The next input is now given to CE by going to step 1.

The recursive nature of CE can be noted by examining the flow chart. The entire routine (step 1) is called recursively by step 3 because of algorithm 1 and by step 5 because of image elaboration.

Also note that step 5 calls step 3 in matching an input to a terminal node.

CHAPTER VI

THE EXAMPLE TASKS

CE must learn some basic information before attempting a more difficult task. First, we describe the basic learning experiment and then three more complex tasks: 1) a concept learning task similar to those in Hunt [8], 2) a paired associate task, and 3) a geometry analogy problem (see Evans [6]). The latter task was not simulated on the computer due to the additional programming it required.

6.1 Basic Information

The example tasks given to CE involve processing of compound objects. In order to learn concepts about compound objects (in a reasonable way), certain elementary concepts involving the subparts of the compound objects should be learned first. Thus, to learn concepts about letter strings and to use letter strings the alphabet should be learned. Step a, given below, is to present CE with written letters as exemplars and their names which are atomic symbols. The written letters are simple objects. A typical list of inputs of this type is:

Step a:

<simple object>	<atomic symbols>	<sign>
A	'A'	+
A	'A'	+
b	'A'	-
A	'A'	+
B	'A'	-
A	'A'	+

The encoding of two simple objects in general will be different. Thus, the A's will all be slightly different. However, any two A's should be more similar than an a and an A. The effect of this learning is to discover the key features (attributes) of the different letters. Key features depend not only on the encoding of the different a's but also on how they differ from the encodings of the other letters.

All simple objects have 15 basic features. Seven of these features are relevant for all letters and the other eight are noise features. Thus, typical instances of a B are generated by providing 8 random feature values. The criteria for when a concept has been learned is that one complete cycle through the list produces correct classification for each element on the list.

In the first step 30 members and non-members of the concept of a capital roman A was given. This list was cyclec through twice to complete learning. Of the 30 A's given, 24 were positive instances of the concept 'A' and 6 were negative instances. Table 2 summarizes the number of simple objects presented to CE in order to learn the alphabet.

Table 2 shows that letters which occured early in the position of learning were hardest to learn. This occured because later letters had a discrimination net to pick out the relevant features of inputs, hence, learning was faster. All inputs of Table 2 were given to CE and stored in memory in 30 seconds on the Univac 1107 computer.

Letters are in more than one concept. For example, the written letter A is an 'A', a capital letter, a vowel, etc. Thus, step b gives inputs to CE which have letters as exemplars and a string of letters as a name. For example, in the input A CAP +, CAP is a compound object naming the concept of "capital" letters. In step b, the concept vowel (VOW), capital letter (CAP), and small letter (SML) are learned.

Step b:*

<simple objects>	<name>	<sign>
a	VOW	+
a	VOW	+
A	CAP	+
A	VOW	+
a	SML	+

*We would like to present 'A' VOW + as an input, but 'A' is a print name and cannot be used as an exemplar.

exempler	name	number of inputs on a list	number of '+' instance	number of '-' instance	number of cycles through the list
A	'A'	30	24	6	2
a	'A'	30	24	6	2
B	'B'	30	24	6	2
b	'B'	30	24	6	2
F	'F'	20	16	4	2
f	'F'	20	16	4	2
J	'J'	20	16	4	2
K	'K'	20	16	4	2
X	'X'	10	8	2	2
Y	'Y'	10	8	2	2
Z	'Z'	10	8	2	2

Table 2

A partial list of inputs given to CE in step a. The inputs appear in order of their position in learning.

Since the alphabet has already been learned by CE, much of the time spent in step b was in learning the names. Table 3 summarizes the inputs given to CE for step b. The inputs of Table 3 were learned by CE in 20 seconds on the Univac 1107 computer.

A snapshop picture of the CE tree after learning the basic information is shown in Figure 8. Notice the names of concepts are always thrown back into the tree and learned, unless the name is a print name. For example, CAP is in the tree as a letter triplet not as an exemplar-name pair. Notice also that tests in Figure 8 may have three branches. When the feature of a test does not designate a subpart of the input the test is undefined. Hence, a third branch, which discriminates inputs with different descriptions, may exist. The example tasks which use this basic information are given next.

exempler	name	number of inputs on a list	number of '+' instance	number of '-' instance	number of cycles through the list
A	VOW	10	8	2	1
A	CAP	10	8	2	1
a	SML	10	8	2	1
b	SML	10	8	2	1
B	CAP	10	8	2	1
J	CAP	10	8	2	1
K	CAP	10	8	2	1
Y	CAP	10	8	2	1
Z	CAP	10	8	2	1

Table 3

A partial list of inputs given to CE in step b.

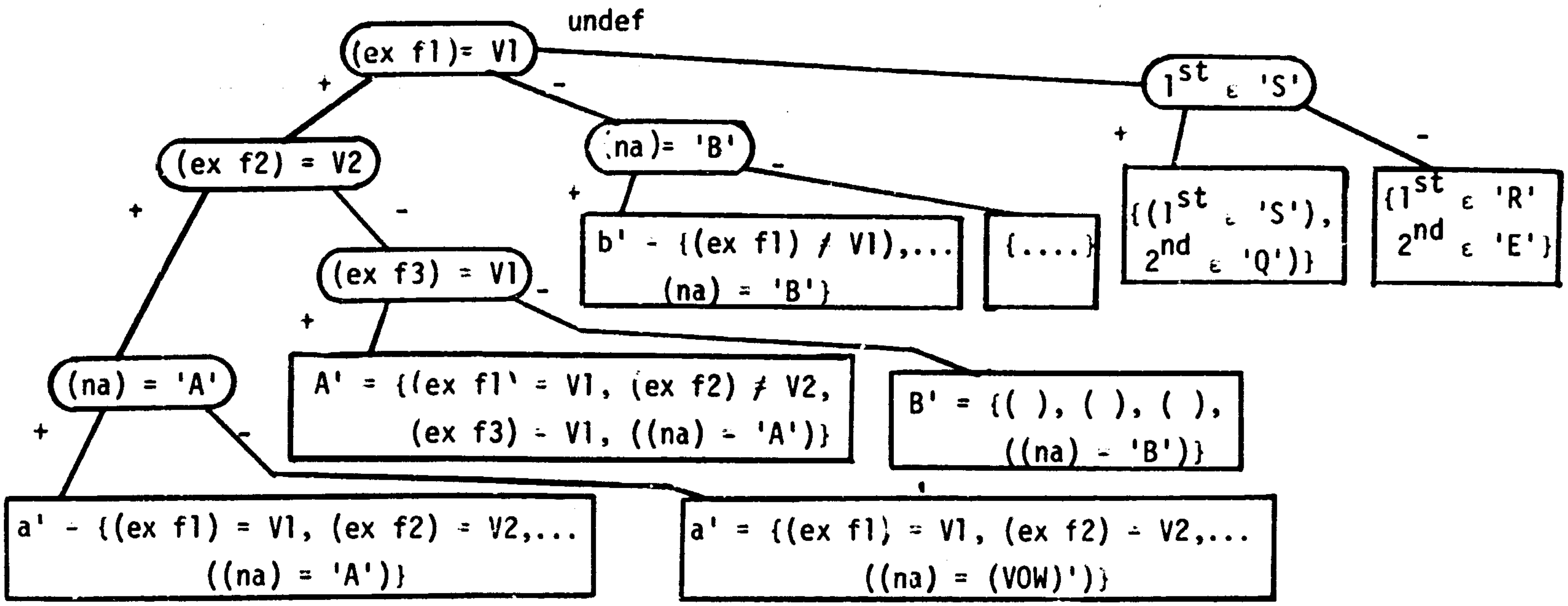


Figure 8

Snapshot picture of CE tree after learning basic information.

6.2 Concept Learning Task

A list of letter strings are given to CE. The exemplars are 'typical' instances of the disjunctive concept, C1, of those three letter words whose third letter is a vowel or whose second letter is a C or both. The growth of the CE tree is shown for each input in sequence, using the learning strategy of Table 1. The inputs are as follows:

	<exemplar>	<name>	<sign>
1.	ABE	C1	+
2.	GCA	C1	+
3.	CEB	C1	-
4.	BCA	C1	+
5.	GDF	C1	-
6.	DCB	C1	+
7.	AGB	C1	-

The actions of the corresponding (serial) inputs are given in order. The actions are numbered according to the number of the corresponding input.

1. The initial image due to input 1 is from throwing the first letter (i.e., A ? +) into the tree and retrieving all names. The name, C1, is also thrown into the CE tree and learned. The initial terminal node is now:

$$\{((ex\ 1^{st})\ \varepsilon\ (CAP)'),\ ((ex\ 1^{st})\ \varepsilon\ 'A'),\ ((ex\ 1^{st})\ \varepsilon\ (VOW)'),\ ((na) = (C1)')\}.$$

2. The tests ((ex 1^{st}) ε 'A') and ((ex 1^{st}) ε (VOW)') are made X-irrelevant (See Section 4.2).
3. All tests are true on the input; ((ex 1^{st}) ε (CAP)') is made Y-irrelevant.
4. Since there are no longer any relevant images at the node, CE decides to elaborate the image by adding the new tests; ((ex 2^{nd}) ε 'C'), ((ex 2^{nd}) ε (CAP)').
5. The test ((ex 2^{nd}) ε (CAP)') is made Y-irrelevant.
6. Nothing is done to the node.
7. Nothing is done to the node.

We now recycle the list of inputs by presenting the same inputs to CE again.

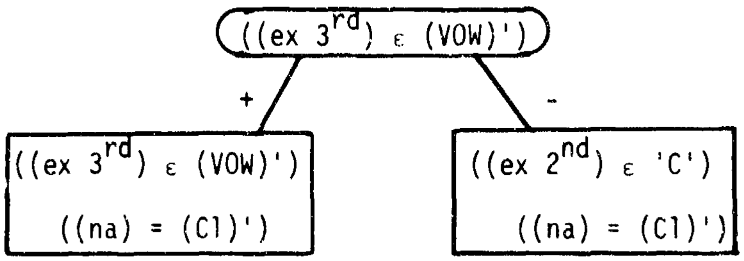
1. The test ((ex 2^{nd}) ε 'C') is made X-irrelevant. The node is now elaborated by adding three new tests; ((ex 3^{rd}) ε 'E'), ((ex 3^{rd}) ε (CAP)') and ((ex 3^{rd}) ε (VOW)').
2. The test ((ex 3^{rd}) ε 'E') is made X-irrelevant.
3. Nothing is done to the node.
4. Nothing is done to the node.
5. Nothing is done to the node.
6. The test ((ex 3^{rd}) ε (VOW)') is made X-irrelevant. The node is now:

{((ex 1^{st}) ε 'A' X), ((ex 2^{nd}) ε 'C' X), ((ex 3^{rd}) ε (VOW)' X),
((ex 1^{st}) ε (CAP)' Y), ((ex 2^{nd}) ε (CAP)' Y), ((ex 3^{rd}) ε 'E' X),
((ex 1^{st}) ε (VOW)' X), ((ex 3^{rd}) ε (CAP)' Y)}

7. The test ((ex 3^{rd}) ϵ (CAP)') is made Y-irrelevant.

Until now CE has tried to describe the concept, C1, conjunctively and has failed. Consequently, CE tries to describe the concept disjunctively by growing the tree. CE must be careful on what test to grow the tree. One of the tests marked X-irrelevant will be used as a new non-terminal node. The criteria is at least two positive inputs must pass the test. In this way CE tries to form a conjunctive concept of a subset of the positive exemplars. Since some positive inputs pass and some fail the test, the test acts to discriminate between conjunctive subparts of a disjunctive concept. This criteria for growing the net is not shown in Table 1 because it is best illustrated with an example. The list of inputs is now recycled again.

1. The tests ((ex 1^{st}) ϵ 'A'), ((ex 1^{st}) ϵ (VOW)'), ((ex 3^{rd}) ϵ 'E'), ((ex 3^{rd}) ϵ (VOW)') are noted as relevant to input 1.
2. The test ((ex 3^{rd}) ϵ (VOW)') is noted as relevant to two positive inputs. Hence, by the growing criteria (above) the test ((ex 3^{rd}) ϵ (VOW)') is the new net discrimination test. The subtree now looks like:



The initial image of input 2 is taken to be those tests marked X-irrelevant that are passed by input 2.

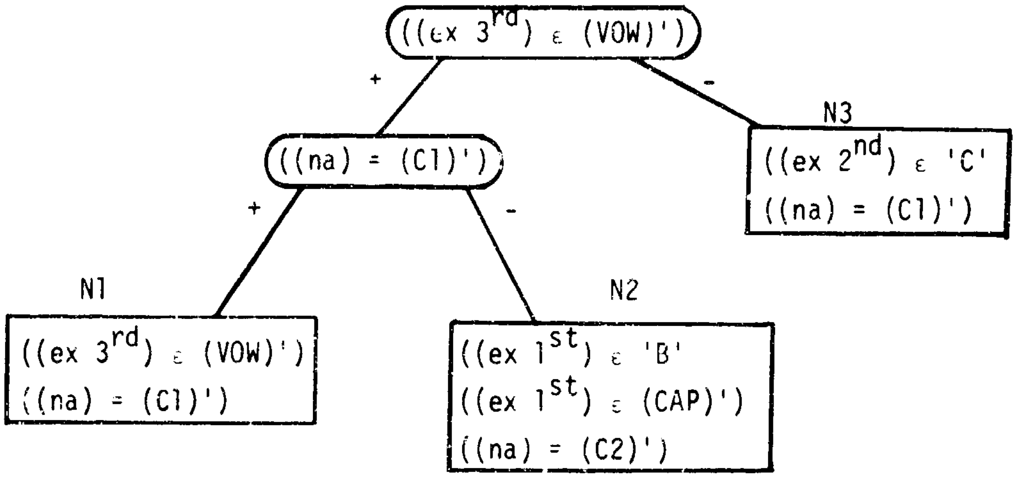
For all remaining inputs, nothing is done to modify the tree; the concept has been learned. The Appendix shows a computer printout of the learned concept, i.e., a subtree of the total memory.

Let us give CE a new list of letter strings. The exemplars of this new list are instances of the conjunctive concept, C2, of those three letter words whose first letter is a B and whose third letter is an A.

The inputs are as follows:

	<exemplar>	<name>	<sign>
1.	BCA	C2	+
2.	BAE	C2	-
3.	BEA	C2	+
4.	CDB	C2	-
5.	ABA	C2	-

1. The initial input sorts to the left node because the test on the first letter is true. Since the exemplar matches the node, a test on the name is the new net discrimination. The tree is now:



An initial image of input 1 is stored in the tree.

2. The input sorts to the node labeled N2. Since the input matches the node all tests are made Y-irrelevant.

3. The input sorts to Node N2. Since all tests are irrelevant, the image is elaborated. The new tests are: ((ex 2^{nd}) ε 'E'), ((ex 2^{nd}) ε (VOW)'), ((ex 2^{nd}) ε (CAP)').

4. The input sorts to node N3 and nothing is done to the tree.

5. The test ((ex 2^{nd}) ε (CAP)') is made Y-irrelevant on node N2.

The input list is now recycled.

1. The tests ((ex 2^{nd}) ε 'E') and ((ex 2^{nd}) ε (VOW)') are marked Y-irrelevant. All the tests are irrelevant so the image is elaborated. The new tests are ((ex 3^{rd}) ε 'A'), and ((ex 3^{rd}) ε (CAP)').

2. The test ((ex 3^{rd}) ε (CAP)') is made Y-irrelevant.

3. Nothing is done to the tree.

4. Nothing is done to the tree.

5. The input matches node N2. Any test marked Y-irrelevant that is false on this input is made relevant. Thus the test ((ex 1^{st}) ε 'B') is made relevant by erasing the Y sign.

The list is recycled again. All inputs are satisfied, hence, the concept has been learned. Node N2 is now:

{((ex 1^{st}) ε 'B'), ((ex 1^{st}) ε (CAP)' Y), ((ex 2^{nd}) ε 'E' X),
((ex 2^{nd}) ε (VOW)' X), ((ex 2^{nd}) ε (CAP)' Y), ((ex 3^{rd}) ε 'A'),
((ex 3^{rd}) ε (CAP)' Y)}.

6.3 Paired Associative Task

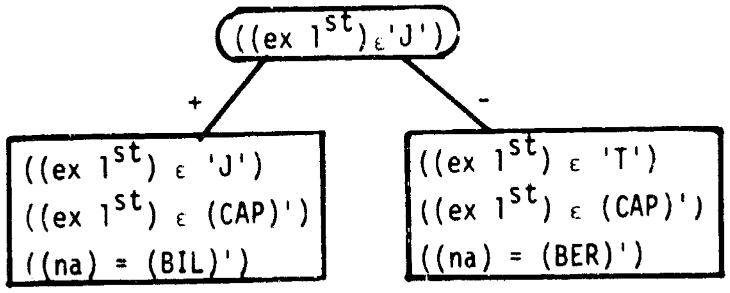
Pairs of letter strings are presented to CE. Hunt [8] points out that this task can be viewed as a pair associative task where exemplars are stimulus and names are responses.

<exemplar or stimuli>	<name or response>	<sign>
JAD	BIL	+
TOD	BER	+
KEP	NIS	+
DAT	REB	+

The first input, JAD BIL +, forms an initial focus image by throwing an examplar subpart into the tree, i.e., J ? +, and retrieving some names. The terminal node is:

$\{((\text{ex } 1^{st})\ \varepsilon\ \text{'J'}),\ ((\text{ex } 1^{st})\ \varepsilon\ (\text{CAP})\text{'}),\ ((\text{na}) = (\text{BIL})\text{'})\}$.

Again, the name, i.e., BIL, is learned. The next input, TOD BER +, is given to CE and a difference is noted in the exemplar, hence, tree modification takes place as shown below:



The next input, KEP NIS +, is sorted to the right node (above) and a difference is detected. The tree is grown using the test $((\text{ex } 1^{st})\ \varepsilon\ \text{'T'})$ as the new net discrimination. The next input, DAT REB +, sorts to the right node and a difference is detected.

The final tree is shown in Figure 9.

Viewing this as a pair associative task, CE will produce responses by asking questions like JAD ? +. Although, in general, such a question may have many answers (due to overlapping concepts), in a paired associative task each such question has a unique answer because each stimulus has a unique response.

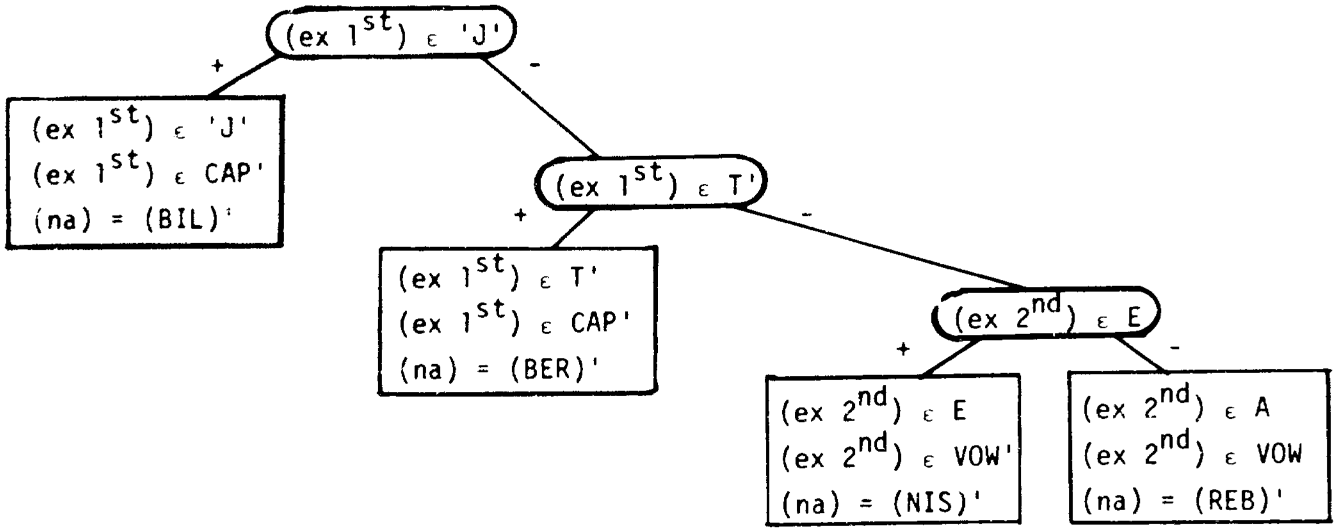


Figure 9

The CE tree after learning the pair associate task.

6.4 Geometry Analogy Task

Until this point the examples have been run on the Univac 1107 computer. The geometry analogy task has not been programmed because of the separate executive needed for learning relational concepts and algorithm 2. CE does not know when to stop describing concepts conjunctively and disjunctively and to start describing concepts as relations.

For the geormetry analogy task, CE should first learn the subparts of the analogy pictures, i.e., the simple geometric figures. Simple geometric figures are encoded as simple objects. Names of the simple objects are letter strings as shown below. In step a, the concept triangle (TR), square (SQ), rectangle (RECT), etc., are learned. The simple geometric figures were not needed for the CE program because the geometry analogy task was not implemented. Sample inputs are given in step a.

Step a:

<simple object>	<name>	<sign>
□	sq	+
□	sq	+
▭	rect	+
△	sq	-
○	sq	-

As an example of learning more complex objects, we shall consider the geometry analogy task*, as typified by the one in Figure 10. The problem is "picture 1 is to picture 2 as picture 3 is to which of the following pictures." The answer, according to most people, is picture 6. The geometry analogy of this problem is that 'the shape of the outer most figure in the left diagram is the same as the shape of the figure in the right diagram." Notice the two figures in the diagram are not identical; we cannot say □ = □ . The two figures are the same because both have the same name (square) with respect to a property (shape). Thus, before CE can learn analogy problems it must have acquired the concept of shape. Since names are stored in CE as concepts, instances that were previously only names can be used as exemplars of new concepts. To this end, CE is told that a square is in the concept shape; a square is a four sided figure (FSF); etc. These inputs were not needed for the CE program because the analogy task was not given to CE. The new executive routine needed for the analogy task is explained in this section. Step b for the geometry analogy task is now given.

*The analogy problem is the same as considered by Evans [6].

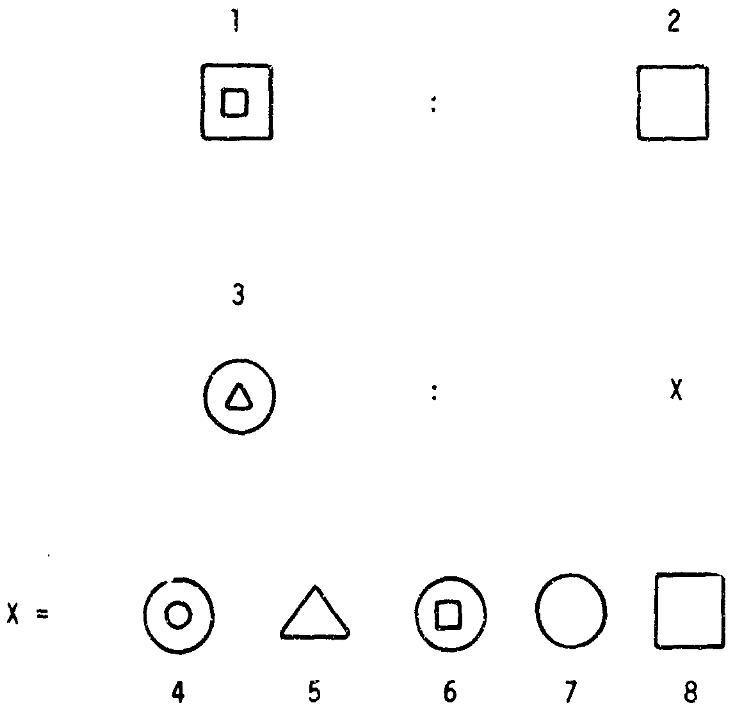


Figure 10

A sample geometry analogy problem from Evans [6]. The problem is "1 is to 2 as 3 is to X." The answer is X = #7.

Step b:

<exemplar>	<name>	<sign>
SQ	SHAPE	+
sq	shape	+
sq	FSF	+
rect	shape	+
Rect	fsf	+

The CE subtree after learning the inputs of step a is illustrated in Figure 11. Note that we have introduced another level in the hierarchy of concepts. That is $\square \in$ sq and sq $\in$ shape; however, $\square \notin$ shape. In general, there will be other concepts at the same "level" as shape, e.g., four sided figures of which sq is also a member.

A difficulty introduced by the geometry analogy task is the need of "relational" concepts. That is, a test like "first is the same as the second." The basic idea is contained in an "extended" question as □ ? + (SHAPE)'. Since the figure has several names, the input will sort to a test on names, say ((na) = (SQ)'). To answer this test we form the question (SQ)' (SHAPE)' ? and throw it into the tree. If the answer is negative, then the test ((na) = (SQ)') is false.

The description of the new test is needed for analogy problems follows. The format of the test is S(<feature$_1$> <feature$_2$> <name>). The logical interpretation of the test is: "There exists a name

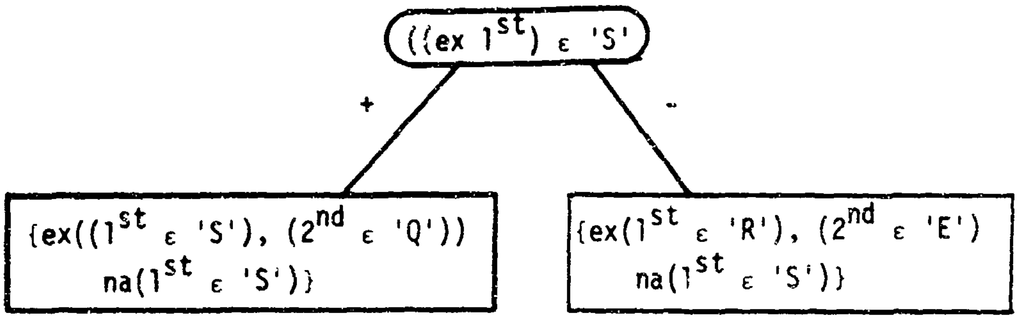


Figure 11

CE subtree after learning step A of the geometry analogy task.

y such that two exemplar subparts, <feature$_1$>, <feature$_2$>, are members of y and y is a member of <name>." The S stands for the predicate, similarity.

The test can be applied to the input

⧈ : □ C5 +.

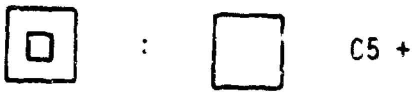
The encoding of the exemplar name pair is given in Figure 12. Thus, the test S((ex left inside)(ex right)(SHAPE)') asks whether the subpart of the exemplar on the left-inside, i.e., □ , is th same as the exemplar on the right, i.e., □ , with respect to shape.

Algorithm 2 for the evaluation of a similarity test is now given. The test S(<feature$_1$> <feature$_2$> <name>) operating on an input is interpreted as "there exists a n_1 such that e_1 n_1 +, e_2 n_1 +, and n_1 n_2 + where e_1 and e_2 are exemplar subparts designated by <feature$_1$> and <feature$_2$> and n_2 is <name>. To find if there exists a n_1, we ask the questions e_1 ? + and e_2 ? +. This implies that non-terminal nodes like ((na) = n) will be encountered. The test ((na) = n) will be true or false depending on the truth of n n_2 ?. In this way the inputs e_1 ? + and e_2 ? + will go to unique terminal nodes, and the names will be retrieved. The similarity test is true if the two names are identical.

As an example of algorithm 2, consider the input,

⧈ : □ C5 +,

and the test, S((ex left inside)(ex right)(SHAPE)'). The first feature, (ex left inside), designates the subpart □ , and the



{ex((left S_s) (right s)
na((1^{st} C) (2^{nd} 5))}

S_s = ((Outside S) (Inside s))
s = ((f1 V1).....(f15 V2))

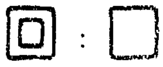
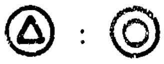
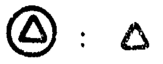
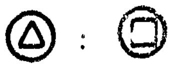
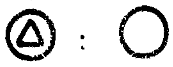
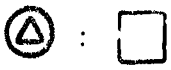
Figure 12

A geometry analogy input and its encoding.

second feature, (ex right), designates □ . The test becomes S(□ , □ . (SHAPE)'). To answer this test the question □ ? + is asked. It goes to a non-terminal node with the test, ((na) = (SQ)'), (See Figure 8). To answer the test we throw SQ' SHAPE' ? into the tree and get the answer +. Thus, ((na) = (SQ)') is true.

Note that the name retrieved in the above example may not be desirable. Suppose the question □ ? + (SHAPE)' reaches a non-terminal node with the test, ((na) = (RECT)'). Since the question, RECT' SHAPE' ? is true, the name, RECT, would be retrieved instead of SQ. However, all squares are rectangles.

An example task of compound (analogy) figures and names is now given. The executive which controls the experiment is explained at the end of the example. The task is for CE to choose which input is most nearly the same as the first input.

	exemplar	name	<sign>
1.		C5	+
2.		C5	?
3.		C5	?
4.		C5	?
5.		C5	?
6.		C5	?

CE will choose input 5 to be true and all other inputs (except 1) to be false; thus, 5 is the answer.

Input 1 is sorted to a new node and a focus image is formed.

Thus, the exemplar subpart ⧈ is thrown into the tree (i.e., ⧈ ? +), but does not match a node. Then the sub-subpart is thrown into the tree, (i.e., □ ? +). The name, SQ' is retrieved. We then throw the name back, (i.e., SQ' ? +) and retrieve SHAPE'. A match of the subparts of the exemplar yields the following terminal node:

{S((ex left outside)(ex right) SHAPE'),
S((ex left inside)(ex right) SHAPE'),
S((ex left outside)(ex left inside) SHAPE'),
((na) = (C5)')}

To complete the analogy task the executive gives the remaining inputs (2 through 6) to CE. If none of the inputs match the terminal node, CE will employ Table 1 to modify the image until exactly one input (2 through 6) matches the node. Thus, the second and third tests are marked irrelevant and input 5 matches the image.

CHAPTER VII

CONCLUSIONS AND OBSERVATIONS

The main feature of this research is the learning of hierarchial concepts, i.e., concepts in terms of other concepts. The concepts can be either conjunctive or disjunctive. CE can generalize upon the instances given to it. In addition, it can correct errors made in generalizing. To the best of our knowledge no other computer program can learn hierarchial concepts, mainly because of the intrinsic dynamics of the problem.

Learning concepts of concepts is closely related to the difficult problem of feature extraction described in Chapter 2. The tests of CE can be viewed as Boolean valued features. For example, for any object the feature, $(ex\ 1^{st}) \in$ 'A', is either true (has a value 1) or false (has a value 0). The problem of learning a concept is the problem of finding the relevant features. In CE this involves finding the concepts possessed by the subparts of an exemplar. In addition these features once "extracted" must be combined with logical connectives (in CE conjunction and disjunction[*]). CE does not consider all possible concepts in "extracting" features; it only considers all those concepts that it has previously learned. This has the good quality that concept

[*] Negation is strange: in order to learn with any level of confidence, the concept $(\exists x\ 1^{st}) \notin$ 'A', at least 25 inputs must be given.

learning improves with time since the repertoire of previously learned concepts is increasing.

Unfortunately, the previously learned concepts come from some concept learning task given to CE. If CE were to do genuine feature extraction, it would have to generate, internally, concept learning tasks as subproblems for some given task. That is, concept learning tasks must be given to CE in a logical order.* Thus, it is that first graders are taught the alphabet before they are taught to read.

To our knowledge the only work that does feature extraction is Uhr and Vossler [16]. However, the features of CE are more general than the templets used in Uhr and Vossler [16]. The templets are fixed in size and are inherently conjunctive. In CE a feature pertains to any concept, either conjunctive, disjunctive or hierarchial, e.g., the disjunctive concept of all a's.

In CE the basis for learning is the question <exemplar> ? +. The answer to this question, <name> , must be retrieved from the CE memory. This process is how CE differs from other work. The implied organization** of other concept learners (e.g., Banerji [1], Hunt [8], Sebestyen [13]) is to look up the name of a concept in a table to find a pointer to the description of the concept. Thus, to

*CE will learn any concept; however, if the correct subconcepts are not learned then learning will be difficult.

**The organization is not stated because they do not have in mind a memory full of concepts. It appears to us that the table look up method is indicated.

answer the question <exemplar> ? + such a program must start at the head of a table and apply the description of the first concept to the exemplar, then the description of the second concept, etc. Since each object may be in several concepts, the search cannot stop with a match, but must go through the entire table. In CE an input is sorted on the exemplar to a particular terminal, and the name is retrieved. If an exemplar is contained in several concepts, then a test on name occurs and a limited search is necessary. However, *never* does CE search the entire memory. Thus, the memory organization of CE is its main advantage because it answers <exemplar> ? + with only a limited search.

One of the primary design criteria is that questions of the form, <exemplar><name> ?, can be answered without search. *Algorithm 1* allows CE to do this. However, the answer may be wrong if the concept was learned incorrectly, and CE will not answer at all if it has not learned sufficient information.

The geometry analogy task at the end of Chapter VI requires relational concepts. Certainly any good concept learner must have this capability. The program that was written learns concepts in terms of conjunctive and disjunctive concepts. *Algorithm 2* is an attempt to retrieve relational concepts without search. However, the whole issue of relational concepts needs more investigation.

The basic difficulty is that identity is meaningless. The statement, □ = □ , does not mean that they are identical but that they both possess the key features of a square. The complication is that they are rectangles and four sided figures also.

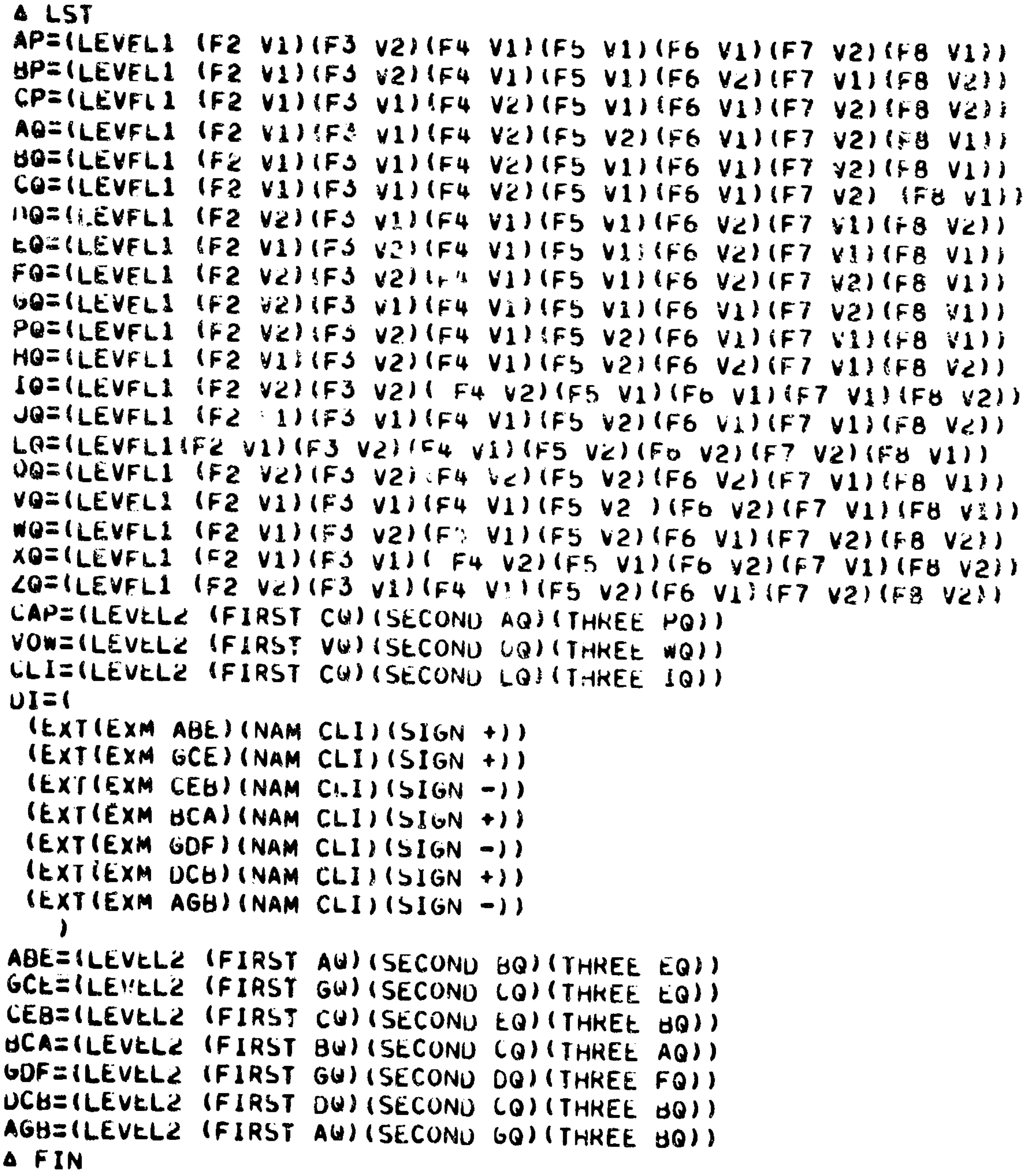
Another difficulty is that CE cannot describe general relational concepts although CE can learn relations. This difficulty can be removed by the technique of Banerji [1]. The set membership test can be extended to form a test that involves several subparts of an object, e.g, ((ex 1st) (ex 3rd)) ε NEXT'. For example, BFC would pass the test because C is next to B in the alphabet. However, the learning of concepts involving arbitrary relations is a difficult problem.

APPENDIX A

COMPUTER INPUT AND OUTPUT

Figure 13 shows some inputs given to the executive. The relevant attributes of simple objects are given. The list of inputs for a particular experiment is illustrated on the list labeled DI. The executive adds random feature values to the description of an input to produce a "typical" input shown in Figure 14.

The concept CLI, corresponding to the concept Cl of Section 6.2, is shown as a subtree of CE in Figure 15. The encoded output is given to make the interpretation easier.



```
Δ LST
AP=(LEVEL1 (F2 V1)(F3 V2)(F4 V1)(F5 V1)(F6 V1)(F7 V2)(F8 V1))
BP=(LEVEL1 (F2 V1)(F3 V2)(F4 V1)(F5 V1)(F6 V2)(F7 V1)(F8 V2))
CP=(LEVEL1 (F2 V1)(F3 V1)(F4 V2)(F5 V1)(F6 V1)(F7 V2)(F8 V2))
AQ=(LEVEL1 (F2 V1)(F3 V1)(F4 V2)(F5 V2)(F6 V1)(F7 V2)(F8 V1))
BQ=(LEVEL1 (F2 V1)(F3 V1)(F4 V2)(F5 V1)(F6 V1)(F7 V2)(F8 V1))
CQ=(LEVEL1 (F2 V1)(F3 V1)(F4 V2)(F5 V1)(F6 V1)(F7 V2) (F8 V1))
DQ=(LEVEL1 (F2 V2)(F3 V1)(F4 V1)(F5 V1)(F6 V2)(F7 V1)(F8 V2))
EQ=(LEVEL1 (F2 V1)(F3 V2)(F4 V1)(F5 V1)(F6 V2)(F7 V1)(F8 V1))
FQ=(LEVEL1 (F2 V2)(F3 V2)(F4 V1)(F5 V1)(F6 V2)(F7 V2)(F8 V1))
GQ=(LEVEL1 (F2 V2)(F3 V1)(F4 V1)(F5 V1)(F6 V1)(F7 V2)(F8 V1))
PQ=(LEVEL1 (F2 V2)(F3 V2)(F4 V1)(F5 V2)(F6 V1)(F7 V1)(F8 V1))
HQ=(LEVEL1 (F2 V1)(F3 V2)(F4 V1)(F5 V2)(F6 V2)(F7 V1)(F8 V2))
IQ=(LEVEL1 (F2 V2)(F3 V2)( F4 V2)(F5 V1)(F6 V1)(F7 V1)(F8 V2))
JQ=(LEVEL1 (F2 V1)(F3 V1)(F4 V1)(F5 V2)(F6 V1)(F7 V1)(F8 V2))
LQ=(LEVEL1(F2 V1)(F3 V2)(F4 V1)(F5 V2)(F6 V2)(F7 V2)(F8 V1))
OQ=(LEVEL1 (F2 V2)(F3 V2)(F4 V2)(F5 V2)(F6 V2)(F7 V1)(F8 V1))
VQ=(LEVEL1 (F2 V1)(F3 V1)(F4 V1)(F5 V2 )(F6 V2)(F7 V1)(F8 V1))
WQ=(LEVEL1 (F2 V1)(F3 V2)(F4 V1)(F5 V2)(F6 V1)(F7 V2)(F8 V2))
XQ=(LEVEL1 (F2 V1)(F3 V1)( F4 V2)(F5 V1)(F6 V2)(F7 V1)(F8 V2))
ZQ=(LEVEL1 (F2 V2)(F3 V1)(F4 V1)(F5 V2)(F6 V1)(F7 V2)(F8 V2))
CAP=(LEVEL2 (FIRST CQ)(SECOND AQ)(THREE PQ))
VOW=(LEVEL2 (FIRST VQ)(SECOND OQ)(THREE WQ))
CLI=(LEVEL2 (FIRST CQ)(SECOND LQ)(THREE IQ))
DI=(
 (EXT(EXM ABE)(NAM CLI)(SIGN +))
 (EXT(EXM GCE)(NAM CLI)(SIGN +))
 (EXT(EXM CEB)(NAM CLI)(SIGN -))
 (EXT(EXM BCA)(NAM CLI)(SIGN +))
 (EXT(EXM GDF)(NAM CLI)(SIGN -))
 (EXT(EXM DCB)(NAM CLI)(SIGN +))
 (EXT(EXM AGB)(NAM CLI)(SIGN -))
   )
ABE=(LEVEL2 (FIRST AQ)(SECOND BQ)(THREE EQ))
GCE=(LEVEL2 (FIRST GQ)(SECOND CQ)(THREE EQ))
CEB=(LEVEL2 (FIRST CQ)(SECOND EQ)(THREE BQ))
BCA=(LEVEL2 (FIRST BQ)(SECOND CQ)(THREE AQ))
GDF=(LEVEL2 (FIRST GQ)(SECOND DQ)(THREE FQ))
DCB=(LEVEL2 (FIRST DQ)(SECOND CQ)(THREE BQ))
AGB=(LEVEL2 (FIRST AQ)(SECOND GQ)(THREE BQ))
Δ FIN
```

Figure 13

Input to the CE Executive

```
1214271(  EXT     121552=         (       SIGN    +       )       121413=         (       EXM     123461=         (       SECOND123566=           (       F1
          V1      )       123543=         (       F9      V2      )       123536=         (       F10     V1      )       123531=         (       F11     V1
          )       123524=         (       F12     V2      )       123517=         (       F13     V2      )       123512=         (       F14     V1      )
          123505=         (       F15     V2      )       123500=         (       F16     V1      )       123561=         (       F2      V1      )       123557
          =       (       F3      V1      )       123555=         (       F4      V2      )       123553=         (       F5      V1      )       123551=
          (       F6      V1      )       123547=         (       F7      V2      )       123545=         (       F8      V1      )       )       123556=
          (       FIRST 123671=           (       F1      V2      )       123646=         (       F9      V1      )       123641=         (       F10     V1
          )       123634=         (       F11     V2      )       123627=         (       F12     V1      )       1236[illegible]2=         (       F13     V2      )
          123615=         (       F14     V2      )       123610=         (       F15     V1      )       123603=         (       F16     V2      )       123664
          =       (       F2      V1      )       123602=         (       F3      V1      )       123660=         (       F4      V2      )       123656=
          (       F5      V2      )       123654=         (       F6      V1      )       123652=         (       F7      V2      )       123650=         (
          F8      V1      )       )       123357=         (       THREE 123462=           (       F1      V1      )       123437=         (       F9      V1
          )       123432=         (       F10     V1      )       123425=         (       F11     V1      )       123420=         (       F12     V2      )
          123413=         (       F13     V1      )       123406=         (       F14     V1      )       123401=         (       F15     V1      )       123374
          =       (       F16     V2      )       123455=         (       F2      V1      )       123453=         (       F3      V2      )       123451=
          (       F4      V1      )       123447=         (       F5      V1      )       123445=         (       F6      V2      )       123443=         (
          F7      V1      )       123441=         (       F8      V1      )       )       )       121411=         (       NAM     123055=         (       FIRST
          123354=         (       F1      V2      )       123331=         (       F9      V2      )       123324=         (       F10     V2      )       123317
          =       (       F11     V1      )       123312=         (       F12     V2      )       123305=         (       F13     V2      )       123300=
          (       F14     V1      )       123273=         (       F15     V1      )       123266=         (       F16     V1      )       123347=         (
          F2      V1      )       123345=         (       F3      V1      )       123343=         (       F4      V2      )       123343=         (       F5
          V1      )       123337=         (       F6      V1      )       123335=         (       F7      V2      )       123333=         (       F8      V1
          )       )       123050=         (       SECOND123251=           (       F1      V2      )       123226=         (       F9      V2      )       123221
          =       (       F10     V1      )       123214=         (       F11     V1      )       123207=         (       F12     V2      )       123202=
          (       F13     V2      )       123175=         (       F14     V1      )       123170=         (       F15     V2      )       123165=         (
          F16     V1      )       123244=         (       F2      V1      )       123242=         (       F3      V2      )       123240=         (       F4
          V1      )       123236=         (       F5      V2      )       123234=         (       F6      V2      )       123232=         (       F7      V2
          )       123230=,        (       F8      V1      )       )       123044=         (       THREE 123146=           (       F1      V1      )       123123
          =       (       F9      V2      )       123116=         (       F10     V2      )       123111=         (       F11     V1      )       123104=
          (       F12     V1      )       123077=         (       F13     V2      )       123072=         (       F14     V1      )       123065=         (
          F15     V1      )       123060=         (       F16     V2      )       123141=         (       F2      V2      )       123137=         (       F3
          V2      )       123135=         (       F4      V2      )       123133=         (       F5      V1      )       123131=         (       F6      V1
          )       123127=         (       F7      V1      )       123125=         (       F8      V2      )       )       )       )
```

Figure 14

An Input to CE From the Executive

```
146413(     146042146033=           (       ND      146014=         (       NT      146010=         (       TT      144340=         (       F2P     145323=
            (       TT      145343=         (       F3P     (       )       145354=         (       FIP     FIRST   )       )       145312=         (       TT
            145324=         (       F3P     1       )       145340=         (       FIP     THREE   )       )       145332=         (       TT      145316=
            (       F3P     L       )       145321=         (       FIP     SECOND)         )       145331=         (       TT      145307=         (       U16431
            )       144324=         (       F3P     V       )       144327=         (       FIP     FIRST   )       )       144343=         (       TT      144320
            =       (       U16431)         143077=         (       F3P     A       )       143102=         (       FIP     SECOND)         )       )       146032
            =       (       FIP     NAM     )       )       )       145251=         (       TRT     146042=         (       TT      146025=         (       F3P
            144331=         (       TT      144302=         (       F3P     U       )       144371=         (       FIP     SECOND)         )       144347=
            (       TT      144332=         (       F3P     #       )       144356=         (       FIP     THREE   )       )       144352=         (       TT
            144324=         (       F3P     V       )       144327=         (       FIP     FIRST   )       )       144343=         (       TT      144320=
            (       U16431)         143077=         (       F3P     A       )       143102=         (       FIP     SECOND)         )       )       146123=
            (       FIP     EXM     THREE   )       )       )       )       146372=         (       ND      146330=         (       NT      146371=         (
            TT      146322=         (       F3P     C       )       146375=         (       FIP     EXM     SECOND)         )       146334=         (       TT
            146353=         (       F2P     145323=         (       TT      145343=         (       F3P     C       )       145354=         (       FIP     FIRST
            )       )       145312=         (       TT      145324=         (       F3P     I       )       14534(=         (       FIP     THREE   )       )
            145332=         (       TT      145316=         (       F3P     L       )       145321=         (       FIP     SECOND)         )       145331=
            (       TT      145307=         (       U16431)         144324=         (       F3P     V       )       144327=         (       FIP     FIRST   )
            )       144343=         (       TT      144320=         (       U16431)         143077=         (       F3P     A       )       143102=         (
            FIP     SECOND)         )       )       146366=         (       FIP     NAM     )       )       )       146341=         (       TRT     145663=
            (       TT      146304=         (       U16431)         146025=         (       F3P     144331=         (       TT      144362=         (       F3P
            U       )       144371=         (       FIP     SECOND)         )       144347=         (       TT      144332=         (       F3P     #       )
            144356=         (       FIP     THREE   )       )       144352=         (       TT      144324=         (       F3P     V       )       144327=
            (       FIP     FIRST   )       )       144343=         (       TT      144320=         (       U16431)         143077=         (       F3P     A
            )       143102=         (       FIP     SECOND)         )       )       146123=         (       FIP     EXM     THREE   )       )       )       )
            )
```

Figure 15

A CE Subtree with Disjunctive Concept CLI

BIBLIOGRAPHY

1. Banerji, R.B., "A Language for the Description of Concepts," *General Systems*, Ludwig, ed., University of Alberta, Edmonton, Canada. Vol IX, 1964, pp. 135-141.

2. Banerji, R.B., "A Language for Pattern Recognition," *Pattern Recognition*, Pattern Recognition Society, to be published.

3. Bower, G.H., and Trabasso, T.R., "Concept Identification," *Studies of Mathematical Psychology*, R.C. Atkinson, Ed., Stanford University Press, 1964.

4. Bruner, J., Goodnow, J., and Austin, G., *A Study of Thinking*, John Wiley and Sons, N. Y., 1956.

5. Ernst, G., "Typed List Structures," Internal Document at Case Western Reserve University, 1967.

6. Evans, T.G., "A Heuristic Program to Solve Geometry Analogy Problems," *Proceedings-Spring Joint Computer Conference*, 1964, pp. 327-338.

7. Feigenbaum, E.A., "The Simulation of Verbal Learning Behaviour," *Computers and Thought*, Feigenbaum and Feldman, ed., McGraw Hill, 1963 pp. 297-309.

8. Hunt, E., Marin, J., and Stone, P., *Experiments in Induction*, Academic Press, N. Y., and London, 1966.

9. Newell, A., "On The Analysis of Human Problem Solving Protocols," Carnegie Institute of Technology, Computer Center Report, 1965.

10. Mucciardi, A., "Adaptive Pattern Recognition Using Non-Linear Elements," Systems Research Center Report SRC-65-A-64-22, Case Institute of Technology, 1964.

11. Pennypacker, J., "An Elementary Information Processor for Object Recognition," Systems Research Center Report SRC-30-I-63-1, Case Institute of Technology, 1963.

12. Quine, W.V., *Word and Object*, The Technical Press of MIT and John Wiley and Sons, N. Y. and London, 1960.

13. Sebestyen, G.S., Decision Making Processes in Pattern Recognition, MacMillian, N. Y., 1962.

14. Simon, H.A., and Feigenbaum, E.A., "Generalization of an Elementary Perciever and Memorizer Machine," IFIPS, 1962, pp. 401-406.

15. Simon, H.A., and Summer, R., "Pattern in Music," Carnegie Institute of Technology, Complex Information Processing paper #104, 1967.

16. Uhr, L., and Vossler, C., "A Pattern Recognition Program That Generates, Evaluates, and Adjusts its own Operators." Computers and Thought, Feigenbaum and Feldman, ed., McGraw Hill, 1963, pp. 251-268.

17. Windeknecht, T.G., "A Theory of Simple Concepts with Applications," Systems Research Center Report SRC-53-A-64-19, Case Institute of Technology, 1964.

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)
Systems Research Center
Case Western Reserve University
Cleveland, Ohio 44106

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3. REPORT TITLE

A MODEL OF CONCEPT LEARNING

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
scientific; interim

5. AUTHOR(S) (First name, middle initial, last name)

Richard H. Sherman

6. REPORT DATE
June 1968

7a. TOTAL NO. OF PAGES
76

7b. NO. OF REFS
17

8a. CONTRACT OR GRANT NO.
AF-AFOSR-67-125A

b. PROJECT NO. 9769-05

c. 61102F

d. 681304

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
AFOSR 68-2084

10. DISTRIBUTION STATEMENT

1. This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES
TECH OTHER

12. SPONSORING MILITARY ACTIVITY
Air Force Office of Scientific Research
Directorate of Information Sciences
Arlington, Virginia 22209

13. ABSTRACT

A learning program designated CE, Concept-EPAM, is described that modifies EPAM through the introduction of a set membership relation. The effects of this extension are considered with respect to methods of storing concept descriptions in memory and methods of specifying learning and retrieval. The learning strategies consist of interactions between image elaboration and tree modification. Implementations ot CE are considered for a concept learning task and a pair associate task. Applicability of CE to a geometry analogy task requiring relational concepts is discussed. The relationship between the learning of concepts of concepts and feature extraction is illustrated.